[illegible]

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/708,021	02/03/2004	Jonas Hogstrom	BORL/0217.00	2020

28653 7590 06/15/2007
JOHN A. SMART
708 BLOSSOM HILL RD., #201
LOS GATOS, CA 95032

EXAMINER	
CHEN, QING	
ART UNIT	**PAPER NUMBER**
2191	
MAIL DATE	**DELIVERY MODE**
06/15/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)
	10/708,021	HOGSTROM ET AL.
	Examiner	Art Unit
	Qing Chen	2191

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.
- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *25 May 2007*.
2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.
3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-63* is/are pending in the application.
4a) Of the above claim(s) _____ is/are withdrawn from consideration.
5)☐ Claim(s) _____ is/are allowed.
6)☒ Claim(s) *1-63* is/are rejected.
7)☐ Claim(s) _____ is/are objected to.
8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.
10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.
Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).
Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).
11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).
a)☐ All b)☐ Some * c)☐ None of:
1.☐ Certified copies of the priority documents have been received.
2.☐ Certified copies of the priority documents have been received in Application No. _____.
3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).
* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

1. This Office action is in response to the amendment filed on May 25, 2007.

2. **Claims 1-63** are pending.

3. **Claims 4, 9, 17, 18, 20, 22, 26, 31, 39, 40, 42, 43, 45, 50, 58, 59, 61, and 63** have been amended.

4. The objections to the oath/declaration are withdrawn in view of Applicant's arguments.

5. The objections to the specification are withdrawn in view of Applicant's amendments to the specification.

6. The objections to Claims 1-63 are withdrawn in view of Applicant's amendments to the claims.

7. The 35 U.S.C. § 112, second paragraph, rejections of Claims 4, 17-20, 26, 39, 40-42, 45, 46, and 58-61 are withdrawn in view of Applicant's amendments to the claims.

8. The 35 U.S.C. § 101 rejections of Claims 22 and 43-63 are withdrawn in view of Applicant's amendments to the claims. However, the 35 U.S.C. § 101 rejections of Claims 23-42 are maintained in view of Applicant's arguments and further explained below.

Response to Amendment

Claim Objections

9. **Claims 2-20, 22, 24-42, and 63** are objected to because of the following informalities:

- **Claims 2-20** recite the statutory category of invention "The method." Applicant is advised to change this statutory category of invention to read "The improved method" for the purpose of providing it with proper explicit antecedent basis.

- **Claims 22 and 63** contain a typographical error: A hyphen (-) should be added between the words "computer" and "readable." Applicant is advised to make the correction in order to keep the claim language consistent throughout the claims.
- **Claims 24-42** recite the statutory category of invention "The system." Applicant is advised to change this statutory category of invention to read "The improved system" for the purpose of providing it with proper explicit antecedent basis.

Appropriate correction is required.

Claim Rejections - 35 USC § 112

10. The following is a quotation of the second paragraph of 35 U.S.C. 112:

The specification shall conclude with one or more claims particularly pointing out and distinctly claiming the subject matter which the applicant regards as his invention.

11. **Claim 46** is rejected under 35 U.S.C. 112, second paragraph, as being indefinite for failing to particularly point out and distinctly claim the subject matter which applicant regards as the invention.

Claim 46 recites the limitation "the programming language." There is insufficient antecedent basis for this limitation in the claim. In the interest of compact prosecution, the Examiner subsequently interprets this limitation as reading "the object oriented programming language" for the purpose of further examination.

Claim Rejections - 35 USC § 101

12. 35 U.S.C. 101 reads as follows:

Whoever invents or discovers any new and useful process, machine, manufacture, or composition of matter, or any new and useful improvement thereof, may obtain a patent therefor, subject to the conditions and requirements of this title.

13. **Claims 23-42** are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter.

Claims 23-42 are directed to systems. However, the recited components of the systems appear to lack the necessary physical components (hardware) to constitute a machine or manufacture under § 101. Therefore, these claim limitations can be reasonably interpreted as computer program modules—software *per se*. Furthermore, the specification discloses that the corresponding apparatus element may be configured in software or firmware *(see Page 15, Paragraph [0034])*. Therefore, the claims are directed to functional descriptive material *per se*, and hence non-statutory.

The claims constitute computer programs representing computer listings *per se*. Such descriptions or expressions of the programs are not physical "things." They are neither computer components nor statutory processes, as they are not "acts" being performed. Such claimed computer programs do not define any structural and functional interrelationships between the computer program and other claimed elements of a computer, which permit the computer program's functionality to be realized. In contrast, a claimed computer-readable medium encoded with a computer program is a computer element, which defines structural and functional interrelationships between the computer program and the rest of the computer, that permits the computer program's functionality to be realized, and is thus statutory. See *Lowry*, 32 F.3d at 1583-84, 32 USPQ2d at 1035.

Claim Rejections - 35 USC § 102

14. The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102 that form the basis for the rejections under this section made in this Office action:

A person shall be entitled to a patent unless –

(b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

15. **Claims 1-6, 12, 15-28, 34, 37, 39-47, 53, and 56-63** are rejected under 35 U.S.C. 102(b) as being anticipated by **Goodwin et al. (US 6,199,195)**.

As per **Claim 1**, Goodwin et al. disclose:

- creating a model describing business objects and rules of the application *(see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML). ")*;

- creating source code for the application, including representing the model within the source code itself *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ")*;

- compiling the source code into an executable application *(see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ... ")*;

- running the executable application on a target computer in conjunction with a run-time framework that provides services to the executable application *(see Column 15: 45-65, "The data server 332 operates at run time ... " and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services. ")*; and

- while the executable application is running, reconstructing the model from the executable application and making it available to the run-time framework *(see Column 13: 65-67 through Column 14: 1-9, "User's own code can be combined with the generated files to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model. "; Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes. " and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions. " and "The data server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases. ")*.

As per **Claim 2**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the model comprises a Unified Modeling Language (UML) model *(see Column 6: 37-44, "... the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).").*

As per **Claim 3**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the source code is created using a programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 4**, the rejection of **Claim 3** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is an object oriented programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 5**, the rejection of **Claim 3** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that Java™ supports the reflection technique.).*

As per **Claim 6**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application *(see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314.").*

As per **Claim 12**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- after reconstructing the model at run-time, testing integrity of the reconstructed model *(see Column 10: 7-11, "Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model ...").*

As per **Claim 15**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is stored in a cache memory available to the run-time framework *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 16**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- wherein the model is initially created using a modeling tool, and wherein the source code is compiled using a compiler *(see Column 8: 44-58, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308."; Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ... ").*

As per **Claim 17**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- representing information of the model in source code as language constructs *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that the Java™ files represent some information of the model in source code as language constructs.).*

As per **Claim 18**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- representing information of the model in source code as attributes *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or*

C++ files ..." It is inherent that the Java™ files represent some information of the model in source code as attributes.).

As per **Claim 19**, the rejection of **Claim 18** is incorporated; and Goodwin et al. further disclose:

- wherein attributes comprise specifiers to structural code elements *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files contain specifiers to structural code elements.).*

As per **Claim 20**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- representing information of the model in code artifacts that exist expressly for carrying model information in source code *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in code artifacts that exist expressly for carrying model information in source code.).*

As per **Claim 21**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- a computer-readable medium having processor-executable instructions for performing the method of claim 1 *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 22**, the rejection of **Claim 1** is incorporated; and Goodwin et al. further disclose:

- a downloadable set of processor-executable instructions for performing the method of claim 1 stored on a computer-readable medium *(see Column 9: 52-57, "The code generator 330 writes source code objects to a directory and returns a uniform resource locator (URL) identified to a client application 338. An operator of the client application (338) is then required to go to the location identified by the uniform resource locator and download the generated code.")*.

As per **Claim 23**, Goodwin et al. disclose:

- a modeling tool for creating a model describing business objects and rules of the application *(see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*;

- a module for creating source code for the application and representing the model within the source code itself *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*;

- a compiler for compiling the source code into an executable application *(see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ... ")*; and

- a run-time framework that is able to reconstruct the model from the executable application and use it for providing services *(see Column 13: 65-67 through Column 14: 1-9, "User's own code can be combined with the generated files to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model."; Column 15: 45-65, "The data server 332 operates at run time ... " and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services.")*.

As per **Claim 24**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the model comprises a Unified Modeling Language (UML) model *(see Column 6: 37-44, "... the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*.

As per **Claim 25**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the source code is created using a programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ").*

As per **Claim 26**, the rejection of **Claim 25** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is an object oriented programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ").*

As per **Claim 27**, the rejection of **Claim 25** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is one that supports reflection technique, thereby allowing reconstruction of the model at run-time from the executable application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... " It is inherent that Java™ supports the reflection technique.).*

As per **Claim 28**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is employed at run-time to support services that the run-time framework provides to the executable application *(see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data* *server 338 for run time operation) consists of developing a unified model of a business application and storing that model in the schema repository 314.").*

As per **Claim 34**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- a submodule for testing integrity of the reconstructed model *(see Column 10: 7-11, "Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model ...").*

As per **Claim 37**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is stored in a cache memory available to the run-time framework *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 39**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the module for creating source code is able to represent information of the model in source code as language constructs *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in source code as language constructs.).*

As per **Claim 40**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the module for creating source code is able to represent information of the model in source code as attributes *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in source code as attributes.).*

As per **Claim 41**, the rejection of **Claim 40** is incorporated; and Goodwin et al. further disclose:

- wherein attributes comprise specifiers to structural code elements *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files contain specifiers to structural code elements.).*

As per **Claim 42**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the module for creating source code is able to represent information of the model in code artifacts that exist expressly for carrying model information in source code *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in code artifacts that exist expressly for carrying model information in source code.).*

As per **Claim 43**, Goodwin et al. disclose:

- creating a model for developing an application using Unified Modeling Language (UML) technique *(see Column 6: 37-44, "The system and method will also allow developers to generate objects based on a framework of services they author by composing services based on the object templates into objects that support the composed behaviors and methods. This is accomplished through the code generator 210, which interfaces with the unified models 206, which are expressed in a unified modeling language, such as Unified Modeling Language (UML).")*;

- generating source code to implement the model *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...")*;

- amending the source code for storing model information in the source code *(see Column 13: 65-67 through Column 14: 1-9, "Output from the code generator 330 can be combined with other user defined codes to create an application.")*;

- compiling the amended source code into an executable application and running the executable application on the computer system *(see Column 14: 34-40, "Outputs from the code generator 404 include Interface Definition Language (IDL) files 422 (*.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ..."; Column 15: 45-65, "The data server 332 operates at run time ..." and "When deployed within a client application, the data server 332 launches, starts, manages and controls execution of a set of services.")*;

- reconstructing the model from the executable application *(see Column 13: 65-67 through Column 14: 1-9, "User's own code can be combined with the generated files to produce* *a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model."; Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes." and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions." and "The data server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases.")*; and

- making the reconstructed model available for supporting operation of the executable application, including rendering the reconstructed model for display *(see Column 13: 65-67 through Column 14: 1-9, "User's own code can be combined with the generated files to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model."; Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes." and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions." and "The data server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases.").*

As per **Claim 44**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the source code is implemented using a programming language *(see Column 13: 52-55; "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 45**, the rejection of **Claim 44** is incorporated; and Goodwin et al. further disclose:

- wherein the programming language is an object oriented programming language *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ...").*

As per **Claim 46**, the rejection of **Claim 45** is incorporated; and Goodwin et al. further disclose:

- wherein the object oriented programming language is one that supports reflection technique, thereby allowing reconstruction of the model from the executable application *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that Java™ supports the reflection technique.).*

As per **Claim 47**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is employed by a run-time framework to provide services to the executable application *(see Column 16: 65-67 through Column 17: 1, "The first step in preparing to use the data server 338 (i.e., preparing the data server 338 for run time* *operation) consists of developing a unified model of a business application and storing that model in the schema repository 314.").*

As per **Claim 53**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- after reconstructing the model, testing integrity of the reconstructed model *(see Column 10: 7-11, "Any authenticated utility can access the meta-information through the schema server 316 using a queryable interface and/or an Interface Definition Language (IDL) interface of a unified modeling language model ...").*

As per **Claim 56**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the reconstructed model is stored in a cache memory *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 57**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- wherein the model is initially created using a modeling tool, and wherein the amended source code is compiled using a compiler *(see Column 8: 44-58, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308."; Column 14: 34-40, "Outputs from the code generator 404 include Interface*

Definition Language (IDL) files 422 (.idl), which are processed by an idl-to-Java compiler, e.g., Visibroker's IDL2JAVA, for the generation of CORBA ORB services ...").*

As per **Claim 58**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- representing information of the model in source code as language constructs *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in source code as language constructs.).*

As per **Claim 59**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- representing information of the model in source code as attributes *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in source code as attributes.).*

As per **Claim 60**, the rejection of **Claim 59** is incorporated; and Goodwin et al. further disclose:

- wherein attributes comprise specifiers to structural code elements *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files contain specifiers to structural code elements.).*

As per **Claim 61**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- representing information of the model in code artifacts that exist expressly for carrying model information in source code *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ..." It is inherent that the Java™ files represent some information of the model in code artifacts that exist expressly for carrying model information in source code.).*

As per **Claim 62**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- a computer-readable medium having processor-executable instructions for performing the method of claim 43 *(see Column 5: 45-47, "The illustrated system 100 has a processor 102 coupled to a memory 104 ...").*

As per **Claim 63**, the rejection of **Claim 43** is incorporated; and Goodwin et al. further disclose:

- a downloadable set of processor-executable instructions for performing the method of claim 43 stored on a computer-readable medium *(see Column 9: 52-57, "The code generator 330 writes source code objects to a directory and returns a uniform resource locator (URL) identified to a client application 338. An operator of the client application (338) is then required* *to go to the location identified by the uniform resource locator and download the generated code.").*

Claim Rejections - 35 USC § 103

16. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

17. **Claims 7, 8, 29, 30, 48, and 49** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin et al. (US 6,199,195)** in view of **Mullins (US 7,103,600).**

As per **Claim 7**, the rejection of **Claim 1** is incorporated; however, Goodwin et al. do not disclose:

- using reflection, reading metadata associated with the executable application to create a graph of code elements; and
- spanning the graph for re-creating the model based on code elements encountered.

Mullins discloses:

- using reflection, reading metadata associated with the executable application to create a graph of code elements *(see Column 36: 27-40, "Now select the top level node to work with. Notice our use of a CocoProxyM class which 'wrappers' the pure java object model class and* *gives the class the compatibility with CocoBase through java reflection instead of requiring any special interfaces in the java class itself ... ")*; and

- spanning the graph for re-creating the model based on code elements encountered *(see Column 36: 50-67, "Now it is possible to step through each of these objects and tell the retrieved object to be navigated through the loadAllLinks method which does the automatic navigation for that object ... ").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include using reflection, reading metadata associated with the executable application to create a graph of code elements; and spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to easily access or track the overall model or diagram of data objects for an object application model or some of its specific elements *(see Mullins – Column 2: 46-49).*

As per **Claim 8**, the rejection of **Claim 7** is incorporated; however, Goodwin et al. do not disclose:

- as each code element is encountered, reconstructing a corresponding portion of the model.

Mullins discloses:

- as each code element is encountered, reconstructing a corresponding portion of the model *(see Column 30: 21-29, "... these same object code listings could have been hard coded by hand and/or produced from another information source UML/XMI or directly from the* *database schema." and 42-50, "This Second Java Object reflects the data of the first Java Object (JSP Processor Class) and allows for further processing to create additional data types from the reflected data types and to translate backwards to the more limited reflected data types that are displayed by the JSP.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include as each code element is encountered, reconstructing a corresponding portion of the model. The modification would be obvious because one of ordinary skill in the art would be motivated to provide complete support for all Java Data Types that might need to be properly retrieved from a data source and persisted back to that or another data source *(see Mullins – Column 30: 47-50).*

As per **Claim 29**, the rejection of **Claim 23** is incorporated; however, Goodwin et al. do not disclose:

- wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered.

Mullins discloses:

- wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered *(see Column 36: 27-40, "Now select the top level node to work with. Notice our use of a*

CocoProxyM class which 'wrappers' the pure java object model class and gives the class the compatibility with CocoBase through java reflection instead of requiring any special interfaces in the java class itself ..." and 50-67, "Now it is possible to step through each of these objects and tell the retrieved object to be navigated through the loadAllLinks method which does the automatic navigation for that object ...").

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include wherein the run-time framework includes submodules for reading metadata associated with the executable application to create a graph of code elements using reflection, and for spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to easily access or track the overall model or diagram of data objects for an object application model or some of its specific elements *(see Mullins – Column 2: 46-49).*

As per **Claim 30**, the rejection of **Claim 29** is incorporated; however, Goodwin et al. do not disclose:

- wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application.

Mullins discloses:

- wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application *(see Column 30: 21-29, "... these same object code listings could have been hard coded by hand and/or* *produced from another information source UML/XMI or directly from the database schema."* *and 42-50, "This Second Java Object reflects the data of the first Java Object (JSP Processor Class) and allows for further processing to create additional data types from the reflected data types and to translate backwards to the more limited reflected data types that are displayed by the JSP.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include wherein the submodule for spanning is able to reconstruct portions of the model based on corresponding code elements encountered in the executable application. The modification would be obvious because one of ordinary skill in the art would be motivated to provide complete support for all Java Data Types that might need to be properly retrieved from a data source and persisted back to that or another data source *(see Mullins – Column 30: 47-50).*

As per **Claim 48**, the rejection of **Claim 43** is incorporated; however, Goodwin et al. do not disclose:

- using reflection, reading metadata associated with the executable application to create a graph of code elements; and

- spanning the graph for re-creating the model based on code elements encountered.

Mullins discloses:

- using reflection, reading metadata associated with the executable application to create a graph of code elements *(see Column 36: 27-40, "Now select the top level node to work with. Notice our use of a CocoProxyM class which 'wrappers' the pure java object model class and* *gives the class the compatibility with CocoBase through java reflection instead of requiring any special interfaces in the java class itself ... ")*; and

- spanning the graph for re-creating the model based on code elements encountered *(see Column 36: 50-67, "Now it is possible to step through each of these objects and tell the retrieved object to be navigated through the loadAllLinks method which does the automatic navigation for that object ... ").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include using reflection, reading metadata associated with the executable application to create a graph of code elements; and spanning the graph for re-creating the model based on code elements encountered. The modification would be obvious because one of ordinary skill in the art would be motivated to easily access or track the overall model or diagram of data objects for an object application model or some of its specific elements *(see Mullins – Column 2: 46-49).*

As per **Claim 49**, the rejection of **Claim 48** is incorporated; however, Goodwin et al. do not disclose:

- as each code element is encountered, reconstructing a corresponding portion of the model.

Mullins discloses:

- as each code element is encountered, reconstructing a corresponding portion of the model *(see Column 30: 21-29, "... these same object code listings could have been hard coded by hand and/or produced from another information source UML/XMI or directly from the* *database schema." and 42-50, "This Second Java Object reflects the data of the first Java Object (JSP Processor Class) and allows for further processing to create additional data types from the reflected data types and to translate backwards to the more limited reflected data types that are displayed by the JSP.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mullins into the teaching of Goodwin et al. to include as each code element is encountered, reconstructing a corresponding portion of the model. The modification would be obvious because one of ordinary skill in the art would be motivated to provide complete support for all Java Data Types that might need to be properly retrieved from a data source and persisted back to that or another data source *(see Mullins – Column 30: 47-50).*

18. **Claims 9, 31, and 50** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin et al. (US 6,199,195)** in view of **Mullins (US 7,103,600)** as applied to Claims 7, 29, and 48 above, and further in view of **Schloegel et al. (US 2004/0044990)**.

As per **Claim 9**, the rejection of **Claim 7** is incorporated; however, Goodwin et al. and Mullins do not disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel et al. disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques *(see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of <u>Schloegel et al.</u> into the teaching of <u>Goodwin et al.</u> to include wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code *(see <u>Schloegel et al.</u> – Paragraph [0034])*.

As per **Claim 31**, the rejection of **Claim 29** is incorporated; however, <u>Goodwin et al.</u> and <u>Mullins</u> do not disclose:

- wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

<u>Schloegel et al.</u> disclose:

- wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques *(see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted* *by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel et al. into the teaching of Goodwin et al. to include wherein the submodule for spanning is able to traverse the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code *(see Schloegel et al. – Paragraph [0034]).*

As per **Claim 50**, the rejection of **Claim 48** is incorporated; however, Goodwin et al. and Mullins do not disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques.

Schloegel et al. disclose:

- wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques *(see Paragraph [0033], "... the order of entity traversal during code generation could be random, or ordered by type, or sorted by name, or filtered so that only entities with a specific property are traversed, or the graph could be traversed in various other ways such as depth-first traversal.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Schloegel et al. into the teaching of Goodwin et al. to include wherein the spanning step includes traversing the graph using a selected one of depth-first, breadth-first, and ad-hoc traversal techniques. The modification would be obvious because one of ordinary skill in the art would be motivated to provide a framework under which it is possible to reason and/or prove qualities about the generated code *(see Schloegel et al. – Paragraph [0034]).*

19. **Claims 10, 11, 32, 33, 51, and 52** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin et al. (US 6,199,195)** in view of **Baisley (US 6,711,734).**

As per **Claim 10**, the rejection of **Claim 1** is incorporated; however, Goodwin et al. do not disclose:

- detecting a class having a package element; and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Baisley discloses:

- detecting a class having a package element *(see Column 4: 35-39, "... a determination is made as to whether or not the MOF Package contains clustered imports (diamond 33).")*; and
- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model *(see Column 4: 35-39, "If the answer to this inquiry is yes, then a "clusteredImport" Tagged Value is created on the UML Model naming each clustered import (block 34).").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting a class having a package element; and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model *(see Baisley – Column 4: 2-8)*.

As per **Claim 11**, the rejection of **Claim 10** is incorporated; however, Goodwin et al. do not disclose:

- detecting an attribute specifying that a class belongs to the UML package; and
- specifying in the reconstructed model that the class belongs to that UML package.

Baisley discloses:

- detecting an attribute specifying that a class belongs to the UML package *(see Column 4: 60-67 through Column 5: 1-2, "... an inquiry is made as to whether or not MOF class has isSingleton=TRUE (diamond 53).")*; and
- specifying in the reconstructed model that the class belongs to that UML package *(see Column 4: 60-67 through Column 5: 1-2, "If the answer to this inquiry is yes, then an "isSingleton" Tagged Value is created with a value of TRUE on the UML class (block 54).")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model *(see Baisley – Column 4: 2-8).*

As per **Claim 32**, the rejection of **Claim 23** is incorporated; however, Goodwin et al. do not disclose:

- wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Baisley discloses:

- wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model *(see Column 4: 35-39, "... a determination is made as to whether or not the MOF Package contains clustered imports (diamond 33). If the answer to this inquiry is yes, then a "clusteredImport" Tagged Value is created on the UML Model naming each clustered import (block 34).").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include wherein the run-time framework includes submodules for detecting a class having a package element, and for creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model *(see Baisley – Column 4: 2-8).*

As per **Claim 33**, the rejection of **Claim 32** is incorporated; however, Goodwin et al. do not disclose:

- a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package.

Baisley discloses:

- a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package *(see Column 4: 60-67 through Column 5: 1-2, "... an inquiry is made as to whether or not MOF class has isSingleton=TRUE (diamond 53). If the answer to this inquiry is yes, then an "isSingleton" Tagged Value is created with a value of TRUE on the UML class (block 54).")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include a module for detecting an attribute specifying that a class belongs to the UML package, and for specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model *(see Baisley – Column 4: 2-8)*.

As per **Claim 51**, the rejection of **Claim 43** is incorporated; however, Goodwin et al. do not disclose:

- detecting a class having a package element; and

- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model.

Baisley discloses:

- detecting a class having a package element *(see Column 4: 35-39, "... a determination is made as to whether or not the MOF Package contains clustered imports (diamond 33).")*; and

- creating a corresponding Unified Modeling Language (UML) package for the reconstructed model *(see Column 4: 35-39, "If the answer to this inquiry is yes, then a "clusteredImport" Tagged Value is created on the UML Model naming each clustered import (block 34).")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting a class having a package element; and creating a corresponding Unified Modeling Language (UML) package for the reconstructed model. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model *(see Baisley – Column 4: 2-8)*.

As per **Claim 52**, the rejection of **Claim 51** is incorporated; however, Goodwin et al. do not disclose:

- detecting an attribute specifying that a class belongs to the UML package; and
- specifying in the reconstructed model that the class belongs to that UML package.

Baisley discloses:

- detecting an attribute specifying that a class belongs to the UML package *(see Column 4: 60-67 through Column 5: 1-2, "... an inquiry is made as to whether or not MOF class has isSingleton=TRUE (diamond 53).")*; and

- specifying in the reconstructed model that the class belongs to that UML package *(see Column 4: 60-67 through Column 5: 1-2, "If the answer to this inquiry is yes, then an "isSingleton" Tagged Value is created with a value of TRUE on the UML class (block 54).")*.

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Baisley into the teaching of Goodwin et al. to include detecting an attribute specifying that a class belongs to the UML package; and specifying in the reconstructed model that the class belongs to that UML package. The modification would be obvious because one of ordinary skill in the art would be motivated to include all of the metadata for the system in a model *(see Baisley – Column 4: 2-8)*.

20. **Claims 13, 14, 35, 36, 54, and 55** are rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin et al. (US 6,199,195)** in view of **Mutschler, III (US 7,162,462)**.

As per **Claim 13**, the rejection of **Claim 12** is incorporated; however, Goodwin et al. do not disclose:

- ensuring that all classes in the model belong to a common superclass.

Mutschler, III discloses:

- ensuring that all classes in the model belong to a common superclass *(see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler, III – Column 5: 40-43).*

As per **Claim 14**, the rejection of **Claim 13** is incorporated; however, Goodwin et al. do not disclose:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

Mutschler, III discloses:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes *(see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler, III – Column 5: 40-43).*

As per **Claim 35**, the rejection of **Claim 34** is incorporated; however, Goodwin et al. do not disclose:

- a submodule for ensuring that all classes in the model belong to a common superclass.

Mutschler, III discloses:

- a submodule for ensuring that all classes in the model belong to a common superclass *(see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include a submodule for ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler, III – Column 5: 40-43).*

As per **Claim 36**, the rejection of **Claim 35** is incorporated; however, Goodwin et al. do not disclose:

- a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass.

Mutschler, III discloses:

- a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass *(see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include a submodule for automatically constructing a common superclass for those classes when all classes in the reconstructed model do not share a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler, III – Column 5: 40-43).*

As per **Claim 54**, the rejection of **Claim 53** is incorporated; however, Goodwin et al. do not disclose:

- ensuring that all classes in the model belong to a common superclass.

Mutschler, III discloses:

- ensuring that all classes in the model belong to a common superclass *(see Column 5: 19-21, "The persistent object 210 represents a superclass from which the named object 220 and other persistent objects 222 are derived.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include ensuring that all classes in the model belong to a common superclass. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler, III – Column 5: 40-43).*

As per **Claim 55**, the rejection of **Claim 54** is incorporated; however, Goodwin et al. do not disclose:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes.

Mutschler, III discloses:

- if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes *(see Column 5: 8-13, "... the effect of a common superclass can be achieved by adding structure fields or data areas representing the attribute data of the superclass to these structures or data blocks and providing procedures or functions in the rule engine program to access and manipulate them.").*

Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to incorporate the teaching of Mutschler, III into the teaching of Goodwin et al. to include if all classes in the reconstructed model do not share a common superclass, automatically constructing a common superclass for those classes. The modification would be obvious because one of ordinary skill in the art would be motivated to share data and methods relating to the classes' common dynamic behavior over time *(see Mutschler, III – Column 5: 40-43).*

21. **Claim 38** is rejected under 35 U.S.C. 103(a) as being unpatentable over **Goodwin et al. (US 6,199,195)**.

As per **Claim 38**, the rejection of **Claim 23** is incorporated; and Goodwin et al. further disclose:

- wherein the model is initially created using a UML modeling tool *(see Column 8: 44-53, "Shown are a number of modeling tools 302, 304, 306 both data modeling 302 and object modeling 304, 306, defining data within a database 308 or defining objects and relating these objects to the data within the database 308." and "... a plurality of model adapters 310 for defining a translation of the logical models of the modeling tools 302, 304, 406 into unified models, expressed in a unified modeling language, such as Unified Modeling Language (UML).").*

However, Goodwin et al. do not disclose:

- wherein the source code is compiled using a C# compiler.

Official Notice is taken that it is old and well known within the computing art to include compiling source code using a C# compiler. A compiler is an essential software component of an

Integrated Development Environment (IDE) used by computer programmers to develop software, such as C# applications. Therefore, it would have been obvious to one of ordinary skill in the art at the time the invention was made to include wherein the source code is compiled using a C# compiler. The modification would be obvious because one of ordinary skill in the art would be motivated to execute applications written in C#.

Response to Arguments

22. Applicant's arguments filed on May 25, 2007 have been fully considered, but they are not persuasive.

In the remarks, Applicant argues that:

a) The Examiner references paragraph [0034] at page 15 of Applicant's specification as stating that the corresponding apparatus element may be configured in software or firmware. However, Applicant respectfully believes that the Examiner has incorrectly construed this paragraph (and the balance of Applicant's specification) as stating that the elements of Applicant's invention can only be implemented in software. In fact, Applicant's specification expressly states that the elements may be implemented in hardware, software or firmware (or combinations thereof). This is expressly stated, for example, at paragraph [0034] of Applicant's specification as follows: "...the corresponding apparatus element may be configured in hardware, software, firmware or combinations thereof" (Applicant's specification, paragraph [0034], emphasis added). Applicant's specification also describes in detail a computer hardware and software environment in which Applicant's invention may be implemented (Applicant's specification, paragraphs [0036]-[0046]). Moreover, Applicant states that the software (computer-executable instructions) direct operation of a device under processor control, such as a computer (Applicant's specification, paragraph [0165]). As Applicant's claim defines a useful machine or item of manufacture in terms of a hardware or hardware and software combination, Applicant respectfully believes that it defines a statutory product and overcomes the rejection of claims 23-42 under Section 101.

Examiner's response:

a) Examiner disagrees with Applicant's assertion that Claims 23-42 define a statutory product. The 35 U.S.C. § 101 rejections of Claims 23-42 are consistent with the Office's current policies regarding non-statutory subject matter. The recited components of the systems appear to lack the necessary physical components (hardware) to constitute a machine or manufacture under § 101. Furthermore, the Examiner has reasons to believe that the recited components of the systems can be reasonably interpreted as computer program modules—software *per se*, since the specification provides intrinsic evidence of such (*i.e.*, the corresponding apparatus element may be configured in hardware, software, firmware *or* combinations thereof) (emphasis added). Consequently, the claims are construed to cover software under the broadest reasonable interpretation. Therefore, the claims are directed to functional descriptive material *per se*, and hence non-statutory. See 35 U.S.C. § 101 rejections of Claims 23-42 above.

In the remarks, Applicant argues that:

b) Although Goodwin describes generating code from a model, Applicant's review of Goodwin finds no teaching of representing the model within the source code itself as provided in Applicant's specification and claims. Goodwin's system provides instead for storing unified models in a schema repository accessed by a schema server (Goodwin, col. 8, lines 58-62; Fig 3). The schema repository of Goodwin's system is a SQL-compliant database (Goodwin, col. 10, lines 38-41). Thus, Goodwin's approach relies on persisting a design-time representation of the model in a database instead of incorporating the model into the application code itself.

Examiner's response:

b) Examiner disagrees. Goodwin et al. clearly disclose representing the model within the source code itself *(see Column 13: 52-55, "Thus, the code generator 330 can support the creation of, for example, IDL, JAVA or C++ files ... ")*. Note that the generated source code is a representation of the model based on the information stored in the model. Furthermore, although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

In the remarks, Applicant argues that:

c) Goodwin has no comparable teaching of reconstructing the model from the executable application at run time. Instead, Goodwin's system relies on retrieving model information through the schema server by a dataserver (Goodwin, col. 10, lines 63-65). The dataserver of Goodwin's system does not reconstruct the model from executable code. Instead, it simply consumes model information from the schema server. The schema server itself doesn't reconstruct any model. The schema server is fed at design time with the original unified model that was used to generate code and the model is persisted by the schema server. In other words, the dataserver of Goodwin's system simply retrieves a stored copy of the model generated at design time from a repository. This is not equivalent to reconstructing the model from the executable application (compiled code) as provided in Applicant's specification and claims.

Examiner's response:

c) Examiner disagrees. Goodwin et al. clearly disclose reconstructing the model from the executable application at run time *(see Column 13: 65-67 through Column 14: 1-9, "User's own code can be combined with the generated files to produce a resultant code library. The library includes user defined behaviors and support for each of the selected services for the given framework, and for interacting with the particular data model represented in the unified model."; Column 16: 36-64, "The unified models contain mappings about the data sources of each of the classes." and "Each of the query managers that allows modification of information (insert, update or delete) must support distributed transactions." and "The data server 332 employs the unified models to provide the clients 338 periodically updated query-based views of distributed heterogeneous databases.")*. Note that for the created application (executable application at run time) can be combined with user defined code, which define behaviors and support for interacting with a particular data model (reconstructing the model). Furthermore, although the claims are interpreted in light of the specification, limitations from the specification are not read into the claims. See *In re Van Geuns*, 988 F.2d 1181, 26 USPQ2d 1057 (Fed. Cir. 1993).

Note that Applicant did not traverse the Examiner's assertion of Official Notice with regard to Claim 38. Therefore, the "old and well known within the computing art" statement is taken to be admitted prior art because Applicant has failed to traverse the Examiner's assertion of Official Notice (see MPEP § 2144.03).

Conclusion

23. **THIS ACTION IS MADE FINAL.** Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the mailing date of this final action.

Any inquiry concerning this communication or earlier communications from the Examiner should be directed to Qing Chen whose telephone number is 571-270-1071. The Examiner can normally be reached on Monday through Thursday from 7:30 AM to 4:00 PM. The Examiner can also be reached on alternate Fridays.

If attempts to reach the Examiner by telephone are unsuccessful, the Examiner's supervisor, Wei Zhen, can be reached on 571-272-3708. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Any inquiry of a general nature or relating to the status of this application or proceeding should be directed to the TC 2100 Group receptionist whose telephone number is 571-272-2100.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free).

[illegible]

WEI ZHEN
SUPERVISORY PATENT EXAMINER

QC / QC
June 11, 2007